Building Synthetic Simulated Environments for Configuring and Training Multi-Camera Systems for Surveillance Applications

Nerea Aranjuelo[1,2], Jorge García[1], Luis Unzueta[1], Sara García[1], Unai Elordi[1,2], and Oihana Otaegui[1]
[1]*Vicomtech, Basque Research and Technology Alliance (BRTA), San Sebastian, Spain*
[2]*Basque Country University (UPV/EHU), San Sebastian, Spain*
{*naranjuelo, jgarciac, lunzueta, sgarcia, uelordi, ootaegui*}*@vicomtech.org*

Keywords: Simulated Environments, Synthetic Data, Deep Neural Networks, Object Detection, Video Surveillance

Abstract: Synthetic simulated environments are gaining popularity in the Deep Learning Era, as they can alleviate the effort and cost of two critical tasks to build multi-camera systems for surveillance applications: setting up the camera system to cover the use cases and generating the labeled dataset to train the required Deep Neural Networks (DNNs). However, there are no simulated environments ready to solve them for all kind of scenarios and use cases. Typically, 'ad hoc' environments are built, which cannot be easily applied to other contexts. In this work we present a methodology to build synthetic simulated environments with sufficient generality to be usable in different contexts, with little effort. Our methodology tackles the challenges of the appropriate parameterization of scene configurations, the strategies to generate randomly a wide and balanced range of situations of interest for training DNNs with synthetic data, and the quick image capturing from virtual cameras considering the rendering bottlenecks. We show a practical implementation example for the detection of incorrectly placed luggage in aircraft cabins, including the qualitative and quantitative analysis of the data generation process and its influence in a DNN training, and the required modifications to adapt it to other surveillance contexts.

1 INTRODUCTION

In recent years, the irruption of Deep Neural Networks (DNNs) has generated significant gains in a variety of Computer Vision tasks, such as object detection (Hou et al., 2019), object segmentation (Maninis et al., 2019), image captioning and visual relationship detection (Xi et al., 2020) or action recognition (Zhang et al., 2019) to build more sophisticated vision-based systems. DNNs have demonstrated the achievement of excellent results using large-scale supervised learning approaches in which a large amount of labeled data sets are usually required for training. Typically, the more data we have available, the better is their performance. In this context, collecting such amount of data can be challenging specifically for two principal reasons.

First, the compliance with privacy-related regulations in some countries, such as the 'General Data Protection Regulation' (GDPR) of the European Union. The data collection process should put in place appropriate technical and organizational measures to implement the data protection principles, such as data protection or data anonymization.

Second, the time and cost that we need to devote in the data collection process, considering that not only the quantity is important but also the variety and balance to appropriately cover all possibilities during training. In the vast majority of cases, the data cannot be directly extracted from the Internet since we try to solve a particular problem in a specific scenario with a specific camera setup. Thus, custom data sets need to be created. This implies the execution of several actions such as data set specification, camera setup installation, technical preparation, recording protocols, even actors simulating precise guide notes. All these actions together with the well-known problem of labeling data (Mujika et al., 2019) due to the lack of effective tools and/or the annotation complexity make the data set generation process more challenging.

There are several ways to extend training data: (i) augmentation techniques (Shorten and Khoshgoftaar, 2019), (ii) domain adaptation techniques and (Singh et al., 2020) (iii) synthetic data generation (Nikolenko, 2019). Augmentation techniques or domain adaptation approaches can be used for those cases in which we already have some annotated data corresponding to the domain. In case we do not, before we start collecting real data, we could tackle the initial stages of the system's design by generating some synthetic data. This would allow us to train initial versions of DNNs that could help us annotating real data, start applying the other mentioned techniques, and progressively improve the system's reliability (Seib et al., 2020).

As stated in (Nikolenko, 2019), synthetic data can be generated following different kinds of strategies, such as compositing real data, relying on generative models, or using simulated environments. The latter has an additional advantage when building multi-camera systems for surveillance applications: they typically provide virtual cameras that can help setting up the camera system to cover the use cases in the targeted real scenario. This is particularly relevant in scenarios not easily accessible for system designers and/or when the camera positions and lens characteristics are not predefined. However, there are no general simulated environments ready to solve the camera setup and data generation tasks for all kinds of scenarios and use cases. Typically, 'ad hoc' environments are built. On the contrary, a generalist solution should allow:

- Including the required scenario-related graphical assets in an easy manner.
- Configuring context- and use-case-based scenes with user-friendly parameters.
- Capturing images from virtual camera viewpoints quickly, taking into account that the rendering time could be an important bottleneck in this process. These images should include camera-related effects, such as the geometric distortion introduced by the lenses.
- Generating a wide and balanced range of plausible situations of interest, randomly, applying suitable noise to the labeled data for the appropriate training of DNNs.

In order to respond to all these challenges, in this work we present a methodology to build synthetic simulated environments for configuring and training multi-camera systems with sufficient generality to be usable in different surveillance contexts, with little effort. We focus on static systems, i.e., those in which the cameras are visualizing the scene from specific positions. This means that applications involving dynamic systems (e.g., robots that interact with the environment while patrolling areas of interest), are beyond this scope. We show a practical implementation example of this methodology in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. We also compare it to alternative state-of-the-art approaches, including the qualitative and quantitative analysis of the data generation process, and the required modifications to adapt it to another surveillance context. To ensure the suitability of the generated data by our methodology, we train a classification DNN and evaluate its accuracy when trained with real and synthetic images.

The rest of the paper is organized as follows: section 2 describes prior works related to synthetic simulated environments for training intelligent systems; section 3 explains our proposed methodology; section 4 presents the mentioned practical implementation example and experiments; finally, section 5 presents the conclusions and future lines of work.

2 RELATED WORK

The recent survey on synthetic data for deep learning (Nikolenko, 2019) shows that in the last years there has been a shift from static synthetic datasets to interactive simulation environments, grouped in the following categories: (1) outdoor urban environments for learning to drive, (2) indoor environments and (3) robotic and aerial navigation simulators. Current state-of-the-art environments have been built upon *Grand Theft Auto V (GTA V)* (Hurl et al., 2019), *Unity3D* (Saleh et al., 2018; Scheck et al., 2020), *Unreal Engine* (Lai et al., 2018; Shah et al., 2017), *CityEngine* (Khan et al., 2019) or *Blender* (Rajpura et al., 2017), among others.

GTA V is an action-adventure video game with realistic graphics of a large detailed city and surrounding areas from which diverse data can be extracted, involving virtual people, animals, cars, trucks, motorbikes, planes, etc. As stated in (Saleh et al., 2018), the main drawback of video-game-based environments is the limited freedom for customization and control over the scenes to be captured, which makes obtaining a large diversity and good balance of classes difficult, due to the rather complicated procedure required to obtain ground-truth instance-level annotations.

On the contrary, (Saleh et al., 2018) claims that their environment, built upon the game engine *Unity3D*, can be set up by one person in one day. This is very little effort, considering that it allows having access to a virtually unlimited number of annotated images with the object classes of state-of-the-art real urban scene datasets. It captures synthetic images and instance-level semantic segmentation maps at the same time and in real-time, with no human intervention. While generating the data it renders the original textures and shaders of included 3D objects and other automatically created unique ones for their

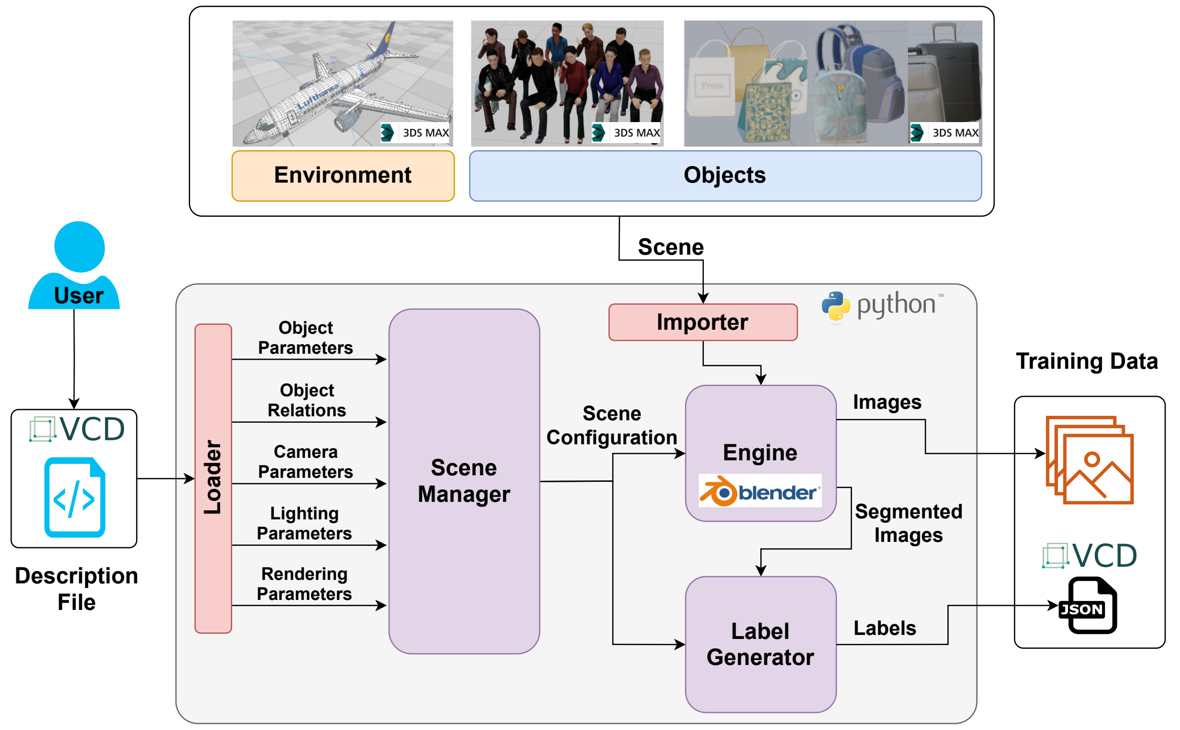


Figure 1: Software architecture of the proposed methodology for creating synthetic training data for vision-based systems.

corresponding instances. *Unity3D* is also the basis of the indoor environment proposed by (Scheck et al., 2020) for the generation of synthetic data for object detection from an omnidirectional camera placed on the ceiling of a room. The 3D assets are generated using the skinned multi-person linear model proposed in (Loper et al., 2015). This work points out that *Unity3D* only provides a camera model for perspective and orthographic projection. They overcome this limitation by combining four perspective cameras following the procedure proposed in (Bourke and Felinto, 2010).

(Lai et al., 2018) proposed a universal dataset and simulator of outdoor scenes such as pedestrian detection, patrolling drones, forest fires, shooting, and more. It is powered by the *Unreal Engine* and leverages the *AirSim* (Shah et al., 2017) plugin for hardware simulation. The TCP/IP protocol is used to communicate with external deep learning libraries. As it is focused on training dynamic systems relying on deep reinforcement learning (Hernandez-Leal et al., 2019), it prioritizes the real-time interactivity of the virtual agents over photorealistic rendering. This differs from static systems for surveillance applications, like those that motivate this work, in which at least for the data generation process, achieving a better rendering quality is more important than the real-time interactivity during training.

CityEngine is a program that allows generating 3D city-scale maps procedurally from a set of grammar rules. It is used in (Khan et al., 2019) as part of their method to generate an arbitrarily large, semantic segmentation dataset reflecting real-world features, including varying depth, occlusion, rain, cloud, and puddle levels while minimizing required effort.

Blender is an open-source 3D graphics software with Python APIs that facilitate loading 3D models and automating the scene rendering. It was used in (Rajpura et al., 2017) to generate a dataset to train a DNN-based detector for recognizing objects inside a refrigerator. It used *Cycles Render Engine* available with *Blender*. This engine is a physically-based path tracer that allows getting photorealistic results, which is beneficial for avoiding a big domain gap between the feature distribution of synthetic and real data domains.

3 PROPOSED METHODOLOGY

Figure 1 shows the architecture of the proposed methodology for generating synthetic data for training deep learning models. The input data of our approach are the 3D assets of the scene and the scene's description file. The output are the synthetic images and the annotations for training DNNs. The principal modules in our approach are: (i) the scene manager which is responsible to load the scene configuration; (ii) the engine which sets up the 3D scene according to the provided configuration and generates the training images by rendering different camera viewpoints and; (iii) the label generator which generates an output file containing the annotations corresponding to the generated images.

Similar to the data collecting process carried out in a real environment, the first step is gathering all the necessary 3D graphical assets to reproduce the scene of interest. Specifically, two different groups of graphical assets are required. The first type contains those graphical assets representing the environment (i.e., the scenario in which we should accomplish the recordings). We assume that such assets belonging to the environment are static since they represent the background. The second group contains those graphical assets representing the dynamic objects of the scene. The combination of the locations of these assets, their poses, sizes, and appearances together with the variations of the lighting sources and the camera properties will provide a wide range of variety for generating our custom training data.

For use cases in which some graphical assets are unavailable, the user should create them using 3D modelling software applications. We use *3DS Max* but it is possible to use other applications such as *Autodesk Maya*, *Lightwave-3D*, *Vectary*, *Blender*, etc. Depending on the complexity, additional plugins such as *Populate* (for *3DS Max*) can alleviate the effort of designing the objects. Working with a very detailed 3D model can become a challenge due to the vast amount of polygons and materials that the render engine has to process. This is directly related to the rendering time of the scene and it could be a troublesome bottleneck in the data flow when a user tries to generate thousands of samples. However, using very detailed 3D assets or more lightweight ones should be considered based on the scope of the use case.

Once all the assets of the scene are designed, our method allows us to configure different camera setups, placing objects at multiple locations and poses, having multiple illumination sources, or configuring the rendering parameters by a minimum user interaction. Furthermore, the user can easily define multiple combinations of parameters for generating different training data samples in a single iteration.

Specifically, the user has to define a configuration file in which the parameterization of the scene is specified in a user-friendly way. Then, a loader interprets the content related to the different entities according to the nature of the parameters (objects, cameras, illumination, rendering). Such information is received by the scene manager, which interprets and handles these data to build the scene configuration for the user-defined sequence. This configuration is used to replicate the target scenes in the 3D synthetic environment, which implies loading the environment and dynamic assets with the corresponding configurations, as well as setting all the cameras, lighting, and rendering parameters to get the desired results. Then, the engine renders the images from the defined camera perspectives. We use *Blender* for this purpose, although the same methodology could be applied using alternative similar programs.

The label generator is in charge of generating the corresponding annotations based on generated segmented images. The user can choose different annotation types depending on the target computer vision task (e.g., object detection or semantic segmentation).

3.1 Scene Management

The synthetic scene is replicated based on the information provided by the user. The user is in charge of the configuration through the description file. For this purpose we adopt the Video Content Description (VCD) structured JSON-schema file format (Vicomtech, 2020). VCD is an open source metadata structure able to describe complex scenes, including annotations and all the needed scene information in a very flexible way. In addition, it allows a very easy and fast user interaction for the files generation process. The VCD format is compatible with the OpenLABEL standard (ASAM, 2020). The configuration file contains the information related to the cameras and lighting setup, the 3D assets in the scene and the relation between them, and the rendering parameters. Modifying something in the scene, such as the position of an object or the camera specifications, is simply done by changing the corresponding field in the configuration file.

3.2 Camera Setup and Lighting Sources

The camera setup controls the way the scenario and the objects are represented in the 2D images. In order to obtain realistic training data, the virtual cameras should simulate the same properties of the sensor and the lens that the expected cameras, which will be installed in the real environment. One of the major benefits of using *Blender* as an engine for creating and rendering the scene, in contrast with others such as *Unity3D*, is the multiple choices of camera models. In particular, we can generate the image projection of the virtual camera by using an orthographic model, a perspective model, or a panoramic model. These three models allow emulating any combination of the image sensor and lens type mounted in a real camera. Table 1 shows the parameters that need to be added to the configuration file to add as many cameras as desired with their corresponding parameters.

Table 1: Camera Parameters

Camera model	Camera model (different distortion types).
Resolution	Output image resolution (pixels).
Position	Sensor position (m) in X, Y and Z axis.
Orientation	Sensor orientation (degrees) in X, Y, and Z axis.
Size	Sensor size (mm).
FOV	FOV of the sensor (degrees).
Focal length	Focal length of the sensor (mm).
Custom params	Extra params defined by the user.

Another important factor affecting the projection of the visual information from 3D to 2D is the lighting of the scene. Depending on the target scenario, the user may want to add light coming from single points which emit light in all directions or from spots with a single direction (e.g., indoor lamps) or an outdoor lighting simulating the sun. Figure 2 shows some example of the mentioned lighting types. Table 2 shows the parameters that should be defined in the configuration file to add as many different light sources as desired to the 3D environment.

Table 2: Lighting Parameters

Light model	Type (point, spot, sun).
Position	Light position (m) in X, Y and Z axis.
Orientation	Light orientation (degrees) in X, Y and Z axis.

The scene configuration is automatically replicated in the 3D environment from the description file. During this step, the scene configuration is exported as a *Blender* project for those cases in which the user wants to interactively explore the camera setup. This way, it can be used as an interactive tool that helps to design a proper setup for a target application. Parameters such as the intrinsic and extrinsic camera parameters, their positions, or even the number of needed cameras can be modified by the user while he/she visualizes the images that would be captured. Simulating a specific set up with no need of physically deploying it, can be of great help to avoid wrong decisions that lead to a not optimal or wrong set up.

This way it may help to define the appropriate number of cameras, their locations, poses, and viewpoints. Thus, the proposed methodology includes the following two bidirectional features:

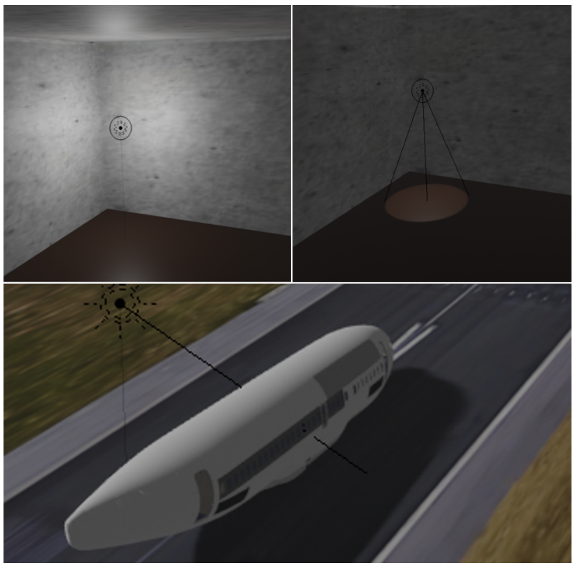


Figure 2: Light sources for a 3D environment: lamp emitting light in all directions (top left) and a single direction (top right), and lighting simulating the sun (bottom).

- VCD2Scene: The user defines the VCD description file and the scene is replicated in the 3D environment including the 3D assets, camera configurations, and lighting sources.
- Scene2VCD: The user loads a 3D scene and after making the desired modifications, he/she exports the new setup to a VCD description file.

3.3 3D Assets in the Scene

Regarding the 3D assets' configuration in the scene, it is also defined in the VCD file. The user can add as many objects as desired to the scene in specific configurations. These objects should belong to the available 3D asset types (e.g., humans, cars). Then the 3D environment simulator interprets this information through the local relation between the assets. The user should have previously defined the relations that will be present in the scene and interactively select the positions they would belong to. For example, if the target application is about detecting abandoned objects in an airport, some local relations that would be necessary could be ”on” or ”below”. These relations would let the user relate different assets for example by describing that certain objects (e.g., a bag, a suitcase) are *on* a desk or *below* the waiting seats. At the same time, the environment simulator would relate these positions with the user-selected positions. In addition, the user defines in the VCD file the time interval when each specific asset is present in the scene in the described configuration. This method provides high flexibility for the user to generate a high variety of configurations in a very fast and user-friendly way.

In order to add a higher degree of variety to the generated data, when each asset is placed on a specific position some random noise is added to slightly perturb its position and orientation. In addition, the user can choose to apply random colors to the 3D assets for including more diversity in the data or on the contrary maintain the original textures.

3.4 Rendering Parameters

The rendering step turns the 3D scene into the output 2D image. The configuration of the rendering affects both the image quality and rendering time. The optimum configuration is closely related to the target task requirements and the available hardware. Therefore, our approach lets the user change the most influential parameters in the configuration file. The parameters that can be modified are shown in Table 3. The user can choose between the available rendering engines (in the case of *Blender* there are three available engines), computing caustics or not, the rendering tile size, the device to be used (CPU or GPU), and the maximum allowed light bounces. When the light hits a surface, it bounces off the surface and hits another one, and then the process is repeated. This is very expensive in terms of rendering time. Decreasing the maximum bounces implies limiting the number of times a ray can bounce before it is killed and therefore, decreasing the time spent computing rays. Depending on the scene and the task, the user can adjust this parameter for getting the desired output.

Table 3: Rendering Parameters

Device	Device used for rendering (CPU/GPU).
Engine	Engine type used for rendering.
Tile	Area of the image considered during the rendering.
Bounces	Maximum light bounces to be applied.
Caustics	Caustics computation.

3.5 Labeled Data Generation

Training machine learning models in a supervised way implies not only collecting the needed data but also the corresponding annotations. Annotating the data is an expensive process for which synthetic data generation can be very helpful thanks to the automatic labels generation. Our approach provides flexible annotations, with different levels of detail depending on the target computer vision task (object detection, object segmentation, or visual relationship detection).

Getting the 3D assets' world coordinates and projecting them to the image plane would be a very efficient way to automatically obtain the annotations. However, it can be observed in Figure 3 (corresponding to a scene in the context explained in the following section) that this could lead to wrong annotations because of occlusions. Different elements of the environment can occlude some objects, so if these occlusions are not considered we would obtain annotations for objects that are not visible in the rendered images. An additional challenge is the camera distortion. Getting accurate annotations implies taking into account and replicating the distortion algorithm used by the chosen 3D graphics software's camera model to obtain the final objects' positions.

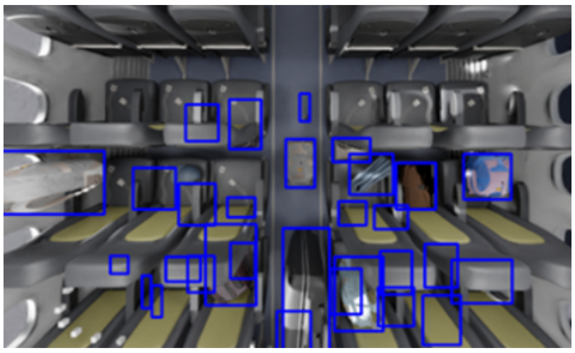


Figure 3: Object annotations based on 3D coordinates can result in some annotated objects occluded by the seats.

We opt for rendering accurate instance-level masks. Each asset in the simulated scene is given a unique ID apart from the object class ID it belongs to. These instance IDs are used to compute an alpha mask per object. These masks are combined in a single segmentation mask. When the data generation starts, the synthetic images are rendered at the same time as the instance-level semantic segmentation maps.

The segmentation masks are used to generate the annotations, which are stored in the output VCD file. This file contains both the input configuration data and the annotations. By default, the masks are used to get the minimum bounding boxes containing each objects' pixels, which are represented in the VCD file by each boxes' corner coordinates. These annotations can be used to train object detectors. However, our approach can be configured to save the objects' segmentation mask too. These data are already in the instance-level masks and are stored as object contours, which are described as polygons in the output file. In addition, the output VCD file contains the description of the relations between the assets in the scene. These data are complemented with the objects' bounding boxes. These annotations can be used to train visual relationship detectors. The annotations need to be parsed to the required format depending on the chosen deep learning framework (e.g., *TensorFlow*, *PyTorch*).

The annotations in the output VCD file are stored per object and framewise. Consequently, each annotation is stored along with its corresponding image path.

4 PRACTICAL CASE AND EXPERIMENTS

In order to validate the proposed methodology, we apply it to a real problem, in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. Currently, the verification of Taxi, Take-off, and Landing (TTL) requirements in aircraft cabins is a manual process. The cabin crew members need to check that all the luggage is correctly placed in each TTL phase. During these phases, the luggage should not be situated in such a way that an emergency evacuation of the aircraft would be delayed or hindered. Figure 4 shows the allowed and not allowed positions for the cabin luggage during TTL. The verification done by the crew members could be automated with the development of a vision-based system. This system would be beneficial in terms of operational efficiency and safety. Developing a system capable of detecting the luggage positions in the cabin entails two main challenges: the design of the camera set up and the generation of suitable data for training the corresponding machine learning models. As stated previously, a 3D environment simulator can be very helpful for these tasks.

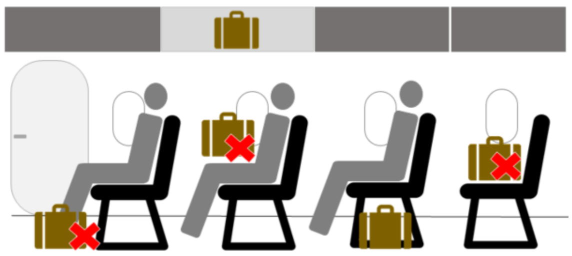


Figure 4: Authorised positions for luggage during TTL.

4.1 3D Assets for the Use Case

To address this problem, the first step of our methodology is the generation of the assets. We first generate all the involved 3D assets for the scene of interest. In this case, we model 22 different object types to simulate typical cabin luggage (e.g., backpacks, magazines, laptops), a cabin model representing a Boeing 737 aircraft (with 19 seats), and a group of human models with different poses and appearances for the seated passengers. The generated 3D assets are shown in Figure 5.

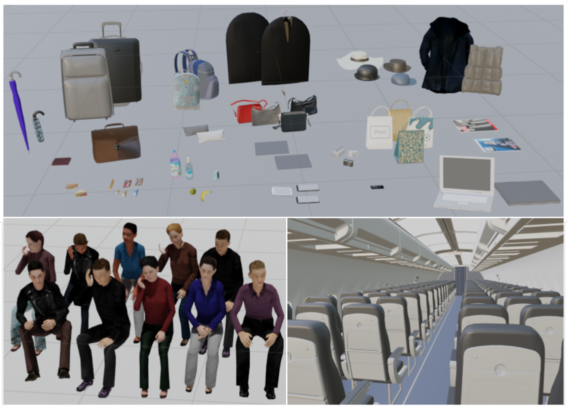


Figure 5: Generated 3D assets for the use case: cabin luggage, passengers, and aircraft cabin.

4.2 Cabin Camera Setup Design

For the camera setup task, our feature VCD2Scene allows loading the aircraft model within some of the modeled 3D assets using an initial configuration file and visualizing directly the 3D scene from the virtual camera viewpoints. The initial setup idea could be to place some cameras on top of the seats to control the luggage in these areas (e.g., backpacks partially below the seats) and some other cameras above the corridor to verify a clear exit. However, how many cameras should be installed? Where should they be to guarantee the system can see every seat without occlusions? What lens parameters should these cameras have?

To answer these questions we interactively change the virtual cameras' extrinsic and intrinsic parameters to check the results we would obtain with each configuration. We move the camera positions and orientations to guarantee that all the regions of interest are captured by the minimum number of cameras.

Figure 6 shows an example of accepted and discarded camera positions for capturing the luggage in the seat areas. At first, we could think the cameras should be on top of the middle seats to capture the status of 6 individual seats (left images). As it can be observed, the passenger seated in the front middle generates an occlusion that does not allow visualizing if he/she has some luggage incorrectly placed. To avoid this blind spot, we test moving the camera towards the windows to capture 4 individual seats (right images). We can see that from this viewpoint there are no occlusion problems, so we opt for this configura-

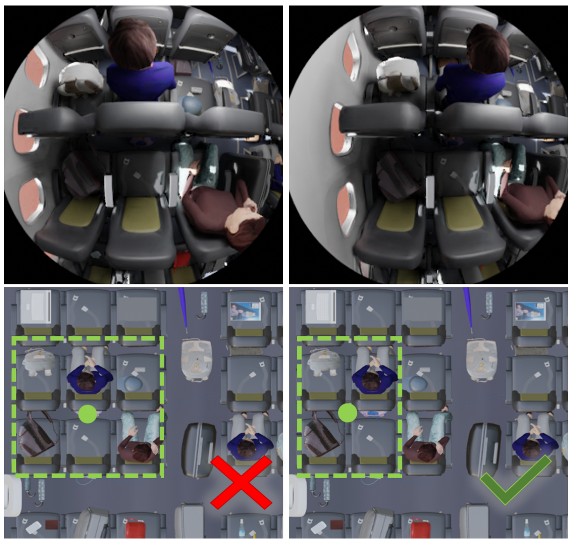


Figure 6: Captured scene from different camera positions. The configuration shown in the left images is discarded because of the occlusions in the front middle seat.

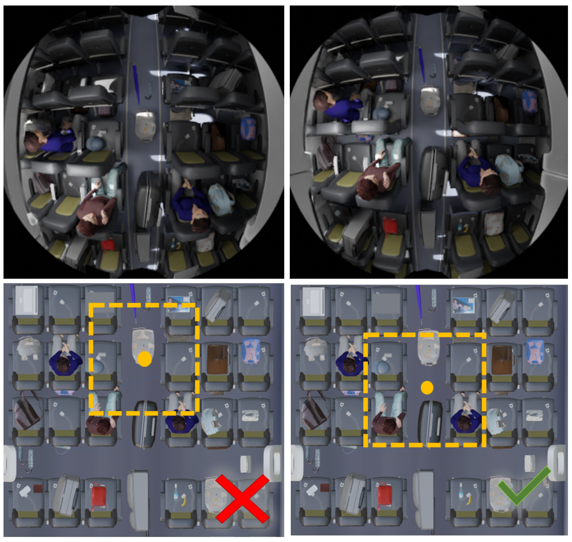


Figure 7: Captured scene from different camera positions. The configuration shown in the left images is discarded because of more occlusions.

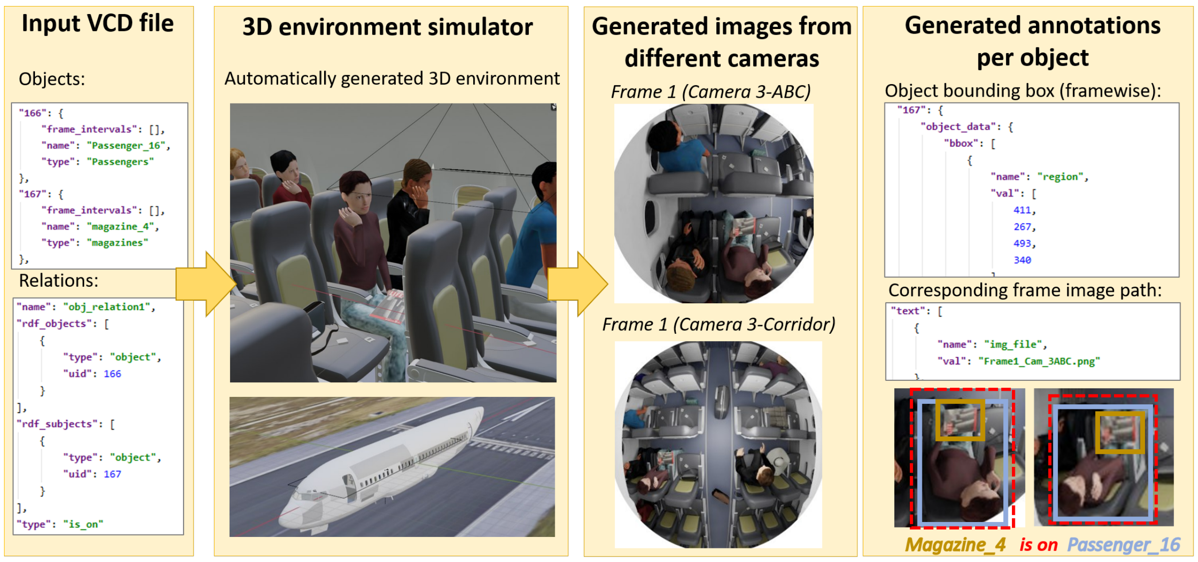
tion. Then, we test setting some cameras in the corridors to verify that area and the seats next to the corridor. Figure 7 shows the tested configurations. Even the corridor space is well visualized in both shown camera positions, the camera should be aligned with the seats (right image) to guarantee the minimum possible occlusions in the feet are of the passengers for as many seats as possible. These tests resulted in a setup design of 20 perspective cameras on top of the seats and 19 on top of the corridor. To guarantee a good visualization of the target areas, we set the parameters of all the cameras to a FOV of 118 degrees, a focal length of 2.13mm, and a sensor size of 4mm.

4.3 Configuration Files Generation

Once we have the final cameras' configuration we export it to an updated VCD configuration file with the Scene2VCD functionality. The file should also contain the 3D assets' configuration so that our environment generator replicates the defined sequences. The situations' variety and the number of object instances of each class can be easily controlled, but depends on the configuration file we use. Generating a balanced dataset is important to guarantee that the trained model does not have a bias for the most common objects in the dataset. Consequently, we define the following requirements for generating the VCD files:

1. An object sample from each object category should be placed at all the possible configurations and places at least once.
2. All the object classes should be present in the generated sequences' frames the same number of times.
3. Samples should show a wide variety of object appearances.

Following these criteria, we generate three VCD files. The first file describes a sequence where each frame contains an object type placed in a specific configuration at all the cabin places (e.g., backpacks on all the seats). The goal of this sequence is to generate enough samples of each object type's appearance in simple configurations (with no interaction of other object types). The second file combines randomly all the objects in all the possible configurations the same number of times. We define a sequence of 500 frames. Each of these frames contains different objects' configurations. The last file follows the same strategy of random combinations but it is focused on the appearance variations. In addition to the default random variations aggregated to the 3D assets position, it enables the color randomization for objects (Section 3.3). Consequently, this file extends the already defined object combinations with new ones that contain objects with a high variety of random appearances. Providing a 3D asset a random color can make some objects look less realistic but increases the variety of samples and can benefit the robustness of the trained models. The strategy used for generating the VCD files is not limited to the current use case, it can be applied to other tasks and scenarios. The defined sequences are summarized in the Table 4.


Input VCD file
Objects:
Relations:
3D environment simulator
Automatically generated 3D environment
Generated images from different cameras
Frame 1 (Camera 3-ABC)
Frame 1 (Camera 3-Corridor)
Generated annotations per object
Object bounding box (framewise):
Corresponding frame image path:
Magazine_4 is on Passenger_16

```
"166": {
    "frame_intervals": [],
    "name": "Passenger_16",
    "type": "Passengers"
},
"167": {
    "frame_intervals": [],
    "name": "magazine_4",
    "type": "magazines"
},
```

```
"name": "obj_relation1",
"rdf_objects": [
    {
        "type": "object",
        "uid": 166
    }
],
"rdf_subjects": [
    {
        "type": "object",
        "uid": 167
    }
],
"type": "is_on"
```

```
"167": {
    "object_data": {
        "bbox": [
            {
                "name": "region",
                "val": [
                    411,
                    267,
                    493,
                    340
```

```
"text": [
    {
        "name": "img_file",
        "val": "Frame1_Cam_3ABC.png"
```

Figure 8: Data generation pipeline example. The input VCD file describes the scene that is replicated by the synthetic 3D environment generator, including present objects and its relations. The tool outputs rendered images and corresponding annotations in VCD format, which correspond to data from different camera perspectives.

Table 4: Summary of the generated data

Sequence	VCD 1	VCD 2	VCD 3	Total
Description	Single object class/frame	Random, balanced	Random appearances	-
Number of frames	45	500	500	1,045
Number of rendered images	1,755	19,500	19,500	40,755
Number of 3D object instances	5,966	33,000	33,000	71,966

4.4 3D Scenes Generation

We use the VCD files to automatically replicate the 3D scenes described in them with no manual intervention. The 3D environment is configured with all the data regarding the cameras, lighting, rendering, and 3D assets to replicate the defined scenes in the cabin. The environment is dynamically configured when the data generation starts.

4.5 Generated Data

Each frame in the sequences is captured from all the defined cameras, so for each frame 39 images are rendered from different camera viewpoints. An output VCD file is generated for each sequence containing the data already in the input file (e.g., camera configurations, relations between objects) and the addition of the output data. The output data include the paths to the generated images and the corresponding bounding box annotations for each object or passenger. Figure 8 shows the data generation process from the input VCD file to the output synthetic data. The left image shows an example of the objects and the relations between them as it is defined in the VCD file. The 3D environment simulator processes this information to configure the 3D scene. It can be seen that the example data ('magazine-4 is on Passenger-14') is replicated in the synthetic 3D world). This scene is captured from the defined cameras to produce the corresponding rendered images, along with the annotations. In the output synthetic images it can be observed that the same 3D assets can be observed from different cameras at the same time. The right images of Figure 8 show an example of the additional information that the output VCD file contains (objects' bounding box coordinates in each rendered frame). This information completes can be used not only for training object detectors but also for training visual relationship detectors.

4.6 Analysis of the Proposed Approach

In this section, we provide a qualitative and quantitative analysis of the proposed methodology's characteristics. Table 5 shows a qualitative comparison of some state-of-the-art approaches to ours.

The environment and 3D assets involved in all the data generation methods are manually modeled with the help of a 3D modeling software or gathered from public repositories. (Khan et al., 2019) also uses data priors such as OpenStreetMap data to model different city environments but still needs manual work to complete and adjust the scene data.

Feature	(Lai et al., 2018)	(Scheck et al., 2020)	(Saleh et al., 2018)	(Rajpura et al., 2017)	(Khan et al., 2019)	Our approach
Environment modeling	Manual	Manual	Manual	Manual	Manual (data priors)	Manual
Scene modifications	Manual	Manual	Manual	Manual	Manual	Automatic
Scene capture	Single-sensor	Single-sensor	Single-sensor	Multisensor	Single-sensor	Multisensor
3D assets relations	Users' interactions	None	None	None	None	Spatial configuration
Annotations	Obj. detection, sem. segmentation, reinforcement learning	Obj. detection, sem. segmentation	Sem. segmentation	Obj. detection	Sem. segmentation, depth estimation	Obj. detection, sem. segmentation, visual relationship detection
Generality	Default scenarios	Limited	Driving scenarios	Limited	Urban driving scenarios	Surveillance scenarios

Table 5: Comparison between state-of-the-art synthetic dataset generation methodologies and our approach.

Once the 3D environment is prepared, if the user wants to change certain scene configurations with our method, such as the light properties, the objects in the scene or the relation between these objects, he/she can define the modified scene in the configuration file using a user-friendly parameterization. Then the new 3D scenario will be automatically replicated. Our approach is the only one that proposes to dynamically configure all the environment when the data generation process starts. Scene modifications can be done in the different approaches but none of them provides a high-level mechanism like ours to vary the environment. Related to the possible 3D assets relations, (Lai et al., 2018) allows adding some limited user interactions. Our approach allows adding spatial location relations between the 3D assets.

Regarding the images capture, all the works propose a single source to capture the scene, except for (Rajpura et al., 2017) and our approach, which allow capturing the same time interval from cameras with different viewpoints.

The data generated by the presented works are oriented to the training of DNNs. Typical output includes annotations for object detection or semantic segmentation tasks, to which (Lai et al., 2018) adds reinforcement learning. Our approach also generates labels suitable for training visual relationship detectors.

One of the main advantages of our approach over the others is its generality and the possibility to adapt it with little effort to new scenarios and tasks in the surveillance field. This feature is very limited to the predefined scenarios in the state-of-the-art works.

Regarding the rendering time, it depends on different factors such as the rendering engine, the complexity of the scene, the number of 3D assets included, the lights configuration, or the polygon number of the modeled objects. For the current aircraft use case, we configure the rendering to be as fast as possible using the parameters in the configuration file maintaining a good output quality. We use the real-time viewport shading rendering for the camera set up design (Section 4.2) so that we can see the modifications' effect interactively. For the data generation, we use the *Cycles Rendering Engine*, which is slower but provides more photorealistic results. Table 6 shows the rendering time (s) computed in different tests to choose the optimum parameters to render a 640x480 pixel image of the cabin environment including 50 3D assets (e.g., suitcases, passengers) apart from the background ones, 16 point lights and outdoors sunlight. We use an Nvidia Tesla T4 GPU. It can be seen that reducing the maximum light bounces from 12 to 1 we need 2.4 seconds less, without noticeable changes in the output image so we adopt only 1 light bounce. Disabling caustics effects neither change the scene appearance and we save an additional 0.1 second. These times are based on using a tile size of 128, but the optimum tile size for our hardware set up is 64. These parameter changes allow us to move from a rendering time of 11.9 seconds to 9.3 seconds. More than 2 seconds per sample in a process where we need to generate thousands of images is a remarkable time-saving. Regarding the rendering time of related state-of-the-art approaches, some works based on game-engines claim to render in real-time (Scheck et al., 2020; Khan et al., 2019). As stated in (Khan et al., 2019), a simplified lighting model allows real-time rendering of massive amounts of geometry with a limited realism. The *Cycles Rendering Engine*, as a physically-based path tracer, allows generating good quality results in a reasonable time. Domain adaptation techniques are important to solve the domain gap that DNNs trained with synthetic data can present. Generating data with a certain degree of realism minimizes the problem to be solved by those techniques. However, *Blender* also has a real-time rendering engine (*Eevee*) that can be used in tasks where the rendering time is considered to be a bigger priority than the data quality.

We apply our methodology to the aircraft use case, but adapting it to another surveillance scenario requires little effort from the user. Once the user collects all the 3D assets of the new environment, the

Max Bounces			Caustics		Tile Size				
12	**6**	**1**	**Yes**	**No**	**16**	**32**	**64**	**128**	**256**
11.9	11.7	9.5	9.5	9.4	16.3	10.8	9.3	9.4	9.5

Table 6: Rendering time in seconds based on different configuration parameters using an Nvidia Tesla T4 GPU.

flexibility of the method allows building a totally new scene. The user interactively designs a suitable camera set up, along with the selection of target places. Then he/she can start gathering variate training data for the considered task based on the configuration files.

4.6.1 DNN Training

In order to see how a DNN would perform with the generated synthetic data and real data, we train the model EfficientNet-B0 (Tan and Le, 2019) to classify whether an image contains a correct or incorrect situation (e.g., cabin luggage correctly or incorrectly placed). We define a region of interest (ROI) for each seat, which will be then classified as correct or incorrect. We replicate the cabin mock-up for capturing also real images from the camera over the seats. Some examples of both real and synthetic ROI images are shown in Figure 9.

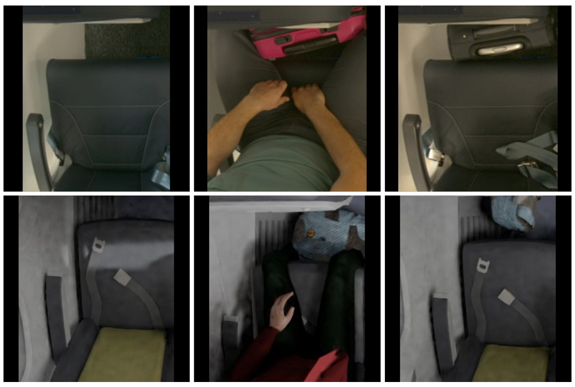


Figure 9: Real (top) and synthetic (bottom) ROI samples for training a classification DNN with correct and incorrect situations.

We do two tests to see the contribution of our synthetic images in the DNN accuracy. First, we train the DNN only with real data and test it on a separated subset of real images. Then, we repeat the training but incorporating the same number of samples from our generated synthetic dataset and test it again on the test real images. We capture and annotate 1,600 real images.

We initialize the DNN with pretrained weights on the ImageNet dataset (Deng et al., 2009) and fine-tune it with our datasets. We train each DNN for 50 epochs with a batch size of 40 and the RMSprop optimizer (Bengio and CA, 2015). The model trained only with real images achieves 88.52% accuracy when classifying the real test samples, while the model which also includes synthetic samples achieves 94.26% accuracy on the same images. Consequently, the positive contribution of the generated synthetic data in the model accuracy confirms the suitability of the generated images for DNNs training.

5 CONCLUSIONS

This work presents a methodology to build simulated 3D environments for configuring and training multi-camera systems with enough generality to be used in different surveillance contexts, with little effort. Our proposal helps in designing an appropriate camera system to cover the target use cases and avoid expensive system setup errors. Once the camera setup is done, our method allows generating a wide range of situations of interest for training DNNs with suitable synthetic data. These situations should be balanced for a successful DNN training. This is guaranteed by the input configuration files, which allow controlling the content of the data with a user-friendly fast scene parameterization. Certain randomness is always added to the scenes in terms of objects' spatial locations or appearances for greater data variability.

We show a practical implementation example of our methodology in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. We design a 39 camera-based system and generate a training dataset of 40,755 images containing a total number of 71,966 object instances in the cabin environment. We follow a data balancing strategy to guarantee the suitability of the generated data based on the configuration files. We also compare our methodology to alternative state-of-the-art approaches. Features such as the generality, flexibility, and multi-camera setting stand out from the features provided by the other approaches. We use a subset of the generated dataset to train a classification DNN jointly with real images. We show that incorporating our synthetic samples to the training dataset boosts the accuracy of the model when tested on real images. Consequently, the images generated by our methodology are suitable for training DNNs.

Future work includes the extension of the methodology to support data generation for action recognition tasks, which requires the possibility of the assets' interaction beyond their configuration and relation. We also plan to continue our experiments about training with the generated synthetic data to ensure their suitability for different computer vision tasks as we keep advancing. In addition, future research will explore the emerging DNN techniques to generate 3D models from 2D images, which may help and speed up the manual process of gathering or designing the 3D assets required for any virtual environment.

ACKNOWLEDGEMENTS

This work has received funding from the Clean Sky 2 Joint Undertaking (JU) under the European Union's Horizon 2020 research and innovation program under grant agreement No. 865162, SmaCS (https://www.smacs.eu/). This paper reflects only the authors' view. The JU is not responsible for any use that may be made of the information it contains.

REFERENCES

ASAM (2020). OpenLABEL. https://www.asam.net/project-detail/scenario-storage-and-labelling/.

Bengio, Y. and CA, M. (2015). Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*.

Bourke, P. and Felinto, D. (2010). Blender and immersive gaming in a hemispherical dome. In *Proc. Int. Conf. on Computer Games, Multimedia and Allied Technology*, volume 1, pages 280–284.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255. Ieee.

Hernandez-Leal, P., Kartal, B., and Taylor, M. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33:750–797.

Hou, Q., Cheng, M., Hu, X., Borji, A., Tu, Z., and Torr, P. H. S. (2019). Deeply supervised salient object detection with short connections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(4):815–828.

Hurl, B., Czarnecki, K., and Waslander, S. L. (2019). Precise synthetic image and LiDAR (PreSIL) dataset for autonomous vehicle perception. *arXiv preprint arXiv:1905.00160*.

Khan, S., Phan, B., Salay, R., and Czarnecki, K. (2019). ProcSy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *Proc. CVPR Workshops*, pages 88–96.

Lai, K.-T., Lin, C.-C., Kang, C.-Y., Liao, M.-E., and Chen, M.-S. (2018). VIVID: Virtual environment for visual deep learning. In *Proc. ACM Int. Conf. on Multimedia (MM)*, pages 1356–1359.

Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. (2015). SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16.

Maninis, K. K., Caelles, S., Chen, Y., Pont-Tuset, J., Leal-Taixé, L., Cremers, D., and Van Gool, L. (2019). Video object segmentation without temporal information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(6):1515–1530.

Mujika, A., Fanlo, A. D., Tamayo, I., Senderos, O., Barandiaran, J., Aranjuelo, N., Nieto, M., and Otaegui, O. (2019). Web-based video-assisted point cloud annotation for ADAS validation. In *Proc. Int. Conf. on 3D Web Technology*, pages 1–9.

Nikolenko, S. I. (2019). Synthetic data for deep learning. *arXiv preprint arXiv:1909.11512*.

Rajpura, P. S., Bojinov, H., and Hegde, R. S. (2017). Object detection using deep CNNs trained on synthetic images. *arXiv preprint arXiv:1706.06782*.

Saleh, F. S., Aliakbarian, M. S., Salzmann, M., Petersson, L., and Alvarez, J. M. (2018). Effective use of synthetic data for urban scene semantic segmentation. *arXiv preprint arXiv:1807.06132*.

Scheck, T., Seidel, R., and Hirtz, G. (2020). Learning from theodore: A synthetic omnidirectional top-view indoor dataset for deep transfer learning. In *Proc. IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 932–941.

Seib, V., Lange, B., and Wirtz, S. (2020). Mixing real and synthetic data to enhance neural network training – a review of current approaches. *arXiv preprint arXiv:2007.08781*.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *Field and Service Robotics*, pages 621–635.

Shorten, C. and Khoshgoftaar, T. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48.

Singh, R., Vatsa, M., Patel, V. M., and Ratha, N., editors (2020). *Domain Adaptation for Visual Understanding*. Springer, Cham.

Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

Vicomtech (2020). VCD - video content description. https://vcd.vicomtech.org/.

Xi, Y., Zhang, Y., Ding, S., and Wan, S. (2020). Visual question answering model based on visual relationship detection. *Signal Processing: Image Communication*, 80:115648.

Zhang, P., Lan, C., Xing, J., Zeng, W., Xue, J., and Zheng, N. (2019). View adaptive neural networks for high performance skeleton-based human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1963–1978.